A

Security at Its Finest: Overview of Cryptography Mechanisms

Awrad Mohammed Ali, University of Central Florida
Neslisah Torosdagli, University of Central Florida
Josiah Wong, University of Central Florida

Cryptography is the field of science which aims to construct systems that can withstand any abuse. The systems are designed in such a way that, even under malicious attacks, the systems can perform their functionalities without any security issues and without revealing any secret information [Goldreich 2004]. Cryptography has been an important field even centuries before modern computers. However, after widespread usage of Internet, it has become an important aspect of our daily life, because personal information such as user names, passwords, credit card information etc. which are required to use modern systems needs to be secure. In this research, we surveyed several papers ranging from classical to more recent ones which become milestones in this field. This paper basically addresses fundamental techniques that have been used in this field, and their applications.

General Terms: Computational Security Algorithms

Additional Key Words and Phrases: Cryptography, Computational Security, One-way Functions, Pseudorandom Generators, One-way permutations, Zero-knowledge

ACM Reference Format:
Awrad Mohammed Ali, Neslisah Torosdagli, Josiah Wong, 2016. Cryptography. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (April 2016), 10 pages.
DOI: 0000001.0000001

1. INTRODUCTION

The original idea of cryptography, to use secret codes to protect privacy of shared information, has been introduced many centuries ago. The need to have a good security code that is hard to break is still one of the most important topics in many modern systems used even in daily life, because private information is required to use most of the modern systems. For instance, credit card information is required to make an online purchase, electronic papers are signed online, or a wide range of private information is shared through the Internet or local Intranet systems, etc. Hence this wide range of private data used on Internet or local Intranet systems needs protection from malicious attempts. Therefore, cryptography being an important field in complexity theory, became an important aspect of daily life. This paper focuses on understanding the fundamental techniques of cryptography. In the following section, we give a historical background about cryptography. Later, we discuss the definition of perfect security and the one-time pad method. Later, we move our discussion to cover one-way functions, pseudorandom generators, zero knowledge proofs, and finally we provide some basic cryptography applications.

2. BACKGROUND

Cryptography is the concatenation of two Greek words meaning “hidden” and “word” respectively, and it covers entire field of secret communication. Secret communication

Author’s addresses: Awrad Mohammed Ali, Neslisah Torosdagli, Josiah Wong, Computer Science Department, University of Central Florida.

DOI: 0000001.0000001

was first invented around 440 B.C. by the Greek ruler Histaeus. Histaeus shaved the head of a slave and tattooed a message on the shaved head. After the growth of the hair, the slave took off to the destination location and delivered the secret message. The recipient uncovered the secret message by shaving slave's head, and replied in the same way.

The need and use of cryptography increased notably during World War II. In this period, promising advances were made both in theory and practice. Alan Turing with a group of British mathematicians broke the German code for sending information to U-boats [Sipser 2006].

In 1970, almost 30 years after World War II, James Ellis realized that it is possible to have secure encryption without sharing secret keys. The British Intelligence Agency implemented this idea in 1973. Intelligence was a closed community, hence almost simultaneously, in Stanford University in US (1976), Diffie and Hellman published a paper on the famous public key cryptography, which immediately yields a public key encryption scheme.

Following the public key encryption, the strongest milestone in cryptography was published by Goldwasser and Micali in 1982, and showed that strong security can be achieved [Goldwasser and Micali 1982]. After 2 years, Goldwasser, Micali and Rivest presented another very strong publication which provides strong security definitions for digital signatures, 2^{nd} version of which is published in 1988 [Goldwasser et al. 1988]. Based on the assumption that integer factorization is hard to invert, they verified how security definitions can be realized. Following Yao's research on next-bit unpredictability [Yao 1982b], the Goldreich-Levin theorem was proved in 1989 [Goldreich and Levin 1989].

Similarly, pseudorandom generators are also being used since the beginning of computing. In 1981, Shamir was the first to deal with intractability of pseudorandomness [Shamir 1983]. Following Shamir, Blum and Micali provided a stronger notion for pseudorandomness [Blum and Micali 1984]. In 1982, Yao showed first protocol for secure realizing any two-party functionality [Yao 1982a], and in 1987, Goldreich, Micali and Wigderson extended this to multi-parties [Goldreich et al. 1987].

Zero-knowledge proofs were invented by Goldwasser, Micali and Rackoff in 1985 [Goldwasser et al. 1985], and they also showed the zero-knowledge proof of the quadratic residue problem. Following this, they proved that if one-way functions do exist, then there is a computational zero-knowledge proof system for every problem in NP [Goldreich et al. 1991].

3. PERFECT SECRECY AND ONE-TIME PAD

A secret key used in the encryption and decryption algorithms is shared when it is required to send a secure message to a specific receiver. In order to have a high security for the exchanged messages, this secret key needs to be well coded so if someone else receives the message he/she will not be able to decode its contents. Practically, this secret key cannot be so short because it would be easily broken using a brute force search. Therefore, researchers found that it would be more secure to have a key length that is as long as the combined messages length. This is known as One-time pad.

One-time pad is a simple idea of encryption that provides perfect security. It is important that every bit of a one-time pad key is used only once to encrypt a bit of the message and later this bit is discarded [Sipser 2006].

It is important to know what researchers mean with the term perfect secrecy in order to understand its requirements. Perfect secrecy is defined as: if we are having an encryption scheme in the form of (E, D), where E is for encryption and D is for decryption. For messages of length m and with a key of length n that satisfy $D_k(E_k(x)) = x$, then (E, D) is perfectly secret if for every pair of messages $x, x' \in \{0,1\}^m$, their distri-

butions are identical [Arora and Barak 2009]. What this definition means is that any eavesdropper should always see a uniform distribution for any siphertext message so it appears as the same distribution as any random message so he will not be able to tell if this message is encrypted or not.

However one-time pad is not a practical solution when we need to securely exchange information in a big size because the perfect secrecy requires the private keys to be as long as the massages. Hence, it is not obvious if such big keys can be exchanged securely among the users. However, if one wants to use a one time pad key shorter than the message the distribution of two messages will not be the same any more [Arora and Barak 2009].

Therefore, the minimal requirement from an encryption is that the eavesdropping would not be able to tell which message from two random messages is encrypted with probability of $\frac{1}{2}$ or better. Hence, the assumption that have been made here is that $P \neq NP$.

On the other hand, in the case of $P = NP$, achieving a perfect secrecy with small key sizes is impossible. This is proven in the following lemma from [Arora and Barak 2009]

LEMMA 3.1.

Suppose that P = NP. Let (E,D) be any polynomial-time computable encryption scheme satisfying $D_k(E_k(x)) = x$ with key shorter than the message. Then, there is a polynomial-time algorithm A such that for every input length m, there is a pair of messages $x_0, x_1 \in \{0,1\}^m$ satisfying:

$$Pr_{\substack{b \in_R \{0,1\} \\ k \in_R \{0,1\}^n}} [A(E_k(x_b)) = b] <= 3/4$$

where n < m denotes the key length for messages of length m.

4. ONE-WAY FUNCTIONS

To overcome the limitations when the length of the messages is long, researchers introduced one-way functions that are used to design secure encryption formula with keys shorter than the messages length. These functions do not give any partial information about the text to a polynomial time eavesdropper.

One-way functions can be defined as functions that are easy to compute but hard to invert in a polynomial time. However, they are only of interest if they are actually exists. Since it is not clear how to prove their existence, researchers assume they are already exist [Arora and Barak 2009]. One-way functions are defined in [Impagliazzo and Luby 1989] as:

Definition 4.1. *f* is somewhat one-way if, for some constant c > 0, for every feasible algorithm ***M***, the inverting probability $Pr[f(x) = f(M(f(x)))]$ when $x \in_U \{0.1\}^n$ is at most $1 - 1/n^c$. We say that *f* is one-way if, for every feasible algorithm ***M***, the inverting probability $Pr[f(x) = f(M(f(x)))]$ when $x \in_U \{0.1\}^n$ is negligible.

One class that is believed to be an example of one-way functions is multiplication functions that split the input $x \in \{0,1\}^n$ to n/2bits numbers and output the result N are examples of one-way functions. Inverting these functions result in integer factorization problem since N is represented by $\log N$ bits which requires exponential time algorithm to invert it. Other examples are RSA and Rabin functions, Levins universal one-way function [Arora and Barak 2009] and AES block ciphers [Daemen and Rijmen 2002].

It is important to notice that it is only guaranteed that one-way function is hard to invert if the input is uniformly distributed and for long enough value of input x [Katz and Lindell 2014].

5. PSEUDORANDOM GENERATORS

Due to the limitations of cryptosystems like one-time pad, a practical cryptosystem must be able to securely encrypt arbitrarily long messages with short key strings. We assert that the existence of one-way permutations implies the existence of a secure pseudorandom generator that can map random inputs of length n to pseudorandom outputs of length polynomial in n. We now proceed to prove that this is indeed the case which would give rise to cryptosystems that use short keys to encode long messages using these pseudorandom generators.

Pseudorandom generators are deterministic algorithms that receive a random input s of length n and output a longer string of length $l(n)$ that seems random to any polynomial-time algorithm known as an adversary. The adversary tries to differentiate between $l(n)$ and another random string of the same length as $l(n)$. The two strings should look random to the adversary if both strings have the same acceptance probability [Håstad et al. 1999].

To prove the security of pseudorandom generators given the existence of one-way permutations, we use an adaptation of the proof in [Arora and Barak 2009] which works as follows: we first show how unpredictability implies pseudorandomness. We then demonstrate via the Goldreich-Levin theorem how pseudorandom generators can map inputs of length n to stretches of length $n+1$. Next, we prove that this result holds for recoverability probabilities of 0.9 and 0.5. Finally, we explain how arbitrarily long stretches are obtained.

5.1. Unpredictability Implies Pseudorandomness

A message is safe from eavesdroppers if it is *unpredictable*; in other words, the ith bit of the message cannot be successfully predicted from the first i-1 bits of the message more than 50% of the time. Thus, a pseudorandom generator is unpredictable; otherwise, there would be an algorithm that could tell random strings from pseudorandom strings which of course is not the case.

We now show that unpredictability implies pseudorandomness as stated here:

— Given polynomial-time computable functions l and G, if G is unpredictable, then G is a pseudorandom generator with stretch $l(n)$.
— If there is an algorithm A that probabilistically behaves differently for input strings from G than for truly random strings, then there is an algorithm B that can guess the ith bit of G's stretch from the first i-1 bits with more than 50% success.

We now prove the second part of the theorem via proof by contradiction. Suppose that G is unpredictable, but not pseudorandom. This means that the probability that an algorithm A outputs a 1 (instead of a 0) given an input string from G is different than that for a truly random input string. Thus, an algorithm B can predict the ith bit of a stretch of length $l(n)$ by running A on a string s consisting of random binary characters (except the first i-1 bits which are known and fixed). If A says the string is random, then B guesses that the ith bit of s is the ith bit of the stretch; otherwise, it says the ith bit of the stretch is the opposite of that of s. Thus, B guesses correctly when A says s is random and the ith bit of s is truly the ith bit of G's stretch or when A says s is not random and the ith bit of s is not the ith bit of the stretch.

Given distributions D_i, $0 \leq i \leq l(n)$, where the strings from D_i have i fixed bits followed by $l(n)$-i random bits, let p_i be the probability that A considers a string from D_i random. The difference between p_0 and $p_{l(n)}$ is non-negligible and is simply the sum of all (p_i - p_{i-1}), $0 \leq i \leq l(n)$, which means that p_i - p_{i-1} is also non-negligible.

The probability that B guesses the ith bit of the stretch correctly is given by this equation: 0.5[Q_1 and Q_2] + 0.5[1 - (Q_1 and not Q_2)] where Q_1 is the probability that

A says *s* is random and Q_2 is the probability that the *i*th bit of *s* is the *i*th bit of the stretch. The probability that (Q_1 and Q_2) is true equals p_i, but the probability that (Q_1 and not Q_2) is true equals $2p_{i-1}$ - p_i because p_{i-1} does not care about the *i*th bit. Thus, *B* correctly guesses the *i*th bit of the stretch with probability 0.5 + p_i - p_{i-1}. This contradicts our assumption that *G* was unpredictable. Therefore, it is enough to prove something is unpredictable in order to prove that it is pseudorandom.

5.2. The Goldreich-Levin Theorem

The next step toward pseudorandom generators with arbitrarily long outputs is proving that pseudorandom generators can derive outputs that are greater in length than their inputs by 1. This is accomplished by the Goldreich-Levin theorem.

The Goldreich-Levin theorem states that given a one-way permutation *f* and a string *r*, no algorithm can determine whether or not the bit-wise AND of *x* and *r* (denoted $x \odot r$) is zero with non-negligibly more than 50% success when run on *f(x)* and *r* alone. By this theorem, a pseudorandom generator *G* when run on inputs *x* and *r* (the combined size of the input being *n* + *n*) can securely output *f(x)*, *r*, and $x \odot r$ (the combined size of the output being *n*+*n*+1).

Suppose for sake of contradiction that there is an algorithm *A* that could determine $x \odot r$ solely from *f(x)* and *r* 100% of the time (contrary to the Goldreich-Levin theorem). Then, an algorithm *B* could invert *f(x)* to get *x* (which contradicts the notion of *f* being one-way) by running *A* *n* times with values of *r* equal to e_i for all $0 \leq i \leq n$. (An *n*-bit string e_i consists of all zeros except the *i*th bit which is one). The output from these *n* runs will be each of the *n* bits of *x*. Thus, *f* is not one-way, a contradiction.

We have just shown that the Goldreich-Levin theorem holds for the impossibility of computing $x \odot r$ 100% of the time. The next two subsections show that the Goldreich-Levin theorem holds for the impossibilities of computing $x \odot r$ with at least 90% success and then with at least 50% success.

5.3. Recovery for Success Probability 0.9

We now adapt the algorithm from the previous section to show that $x \odot r$ cannot be computed with at least 90% success. For sake of contradiction, suppose there is an algorithm *A* that computes $x \odot r$ with at least 90% success. It is no longer possible to rely on values of *r* equal to e_i, $1 \leq i \leq n$, because all of those strings may be among the $2^n/10$ strings for which *A* does not successfully compute $x \odot r$. However, we can still prove that an algorithm *B* can invert *f(x)* to determine *x* with more than 90% success, contradicting the assumption that *f* is one-way.

Since *r* is from the uniform distribution, so is $r \oplus e_i$ for all *n* values of *i*. For all values of *m* (for any given value of *i*), let z_j equal the result of running *A* on *f(x)* and *r* and let z'_j equal the result of running *A* on *f(x)* and $r \oplus e_i$. Then, $z_j \oplus z'_j$ equals the *i*th bit of *x* at least 80% of the time (because the chance of z_j and z'_j being incorrect are less than 10% each). However, *B* will guess that the value of the *i*th bit of *x* equals the most frequent value (the majority value) out of all *m* values of $z_j \oplus z'_j$.

We now show that the majority value is correct more than 90% of the time when *m*=200*n*. For any given value of *i*, $1 \leq i \leq n$, let Z_j, $1 \leq j \leq m$, be 1 if z_j and z'_j are both 1. Let *Z* be the sum of all Z_j. The expected value of any *Z* is at least 0.8*m* (because the expected value of any Z_j is at least 0.8). Since we only need to show that *Z* is expected to take the majority at least 90% of the time (a.k.a. the probability that $Z \leq m/2$ is less than 1-1/(10*n*)), we can just prove the equivalent statement that the probability that the absolute difference between the actual value of *Z* and the expected value of *Z* exceeds 0.8*m*-0.5*m*=0.3*m* is less than 10% (or 1/(10n)). Chebychevs Inequality states that the probability that the actual value of a variable falls outside of *k* standard

deviations from its mean is less than $1/k^2$. If we let m=200n, then the probability that Z is more than $0.3m$ away from its expected value is $1/(0.3m^{1/2})^2$ which is less than $1/(10n)$.

5.4. Recovery for Success Probability 0.5

We now adapt the proof from the previous section to prove the general result of the Goldreich-Levin theorem, that $x \odot r$ cannot be determined from *f(x)* and *r* alone by any algorithm non-negligibly more than 50% of the time. The key is that the m strings for *r* only have to be pairwise independent instead of fully independent to use Chebychevs Inequality. Again, we use proof by contradiction and assume that an algorithm *A* can compute $x \odot r$ from *f(x)* and *r* alone with non-negligibly more than 50% success.

We now show how an algorithm *B*, given value *y=f(x)* and function *f*, will invert *f(x)* to get *x* for some *x* where *A* can successfully compute $x \odot r$ at least 50% of the time. Again, let m=200n and let k be the smallest integer where $m \leq 2^k$-1. Choose k random binary strings s_k and for every j, $1 \leq j \leq m$, let T_j consist of a unique subset of integers between 1 and k. Let r_j be the exclusive-or of all strings s_j in T_j. For all binary strings w of size k, we assume w_t=$x \odot s_t$ and thus z_j (for a given i, $1 \leq i \leq n$, and for all j, $1 \leq j \leq m$) equals the exclusive-or of all w_t in T_j. Also, set z'_j to $x \odot (r_j \oplus e_i)$. Let the guess for the ith bit of *x* be the majority value for $z_j \oplus z'_j$ for $1 \leq j \leq m$. Test if the *x* (the concatenation of all n bits that were guessed) satisfies *y=f(x)* and if so, we have computed *x*. We test all 2^k values of w_t because 2^k = *O(m)*; at least one of these strings will produce the correct *x*.

We now finish this proof by showing that algorithm *B* inverts *x*, contradicting the assumption that *f* is one-way. Define m values of Z_j which are equal to 1 if z_j=x$\odot$ r_j and z'_j=$x \odot (r_j \oplus e_i)$ for one of the n values of i. When w_t produces the correct *x*, we know that z_j=$x \odot$ r_j and thus we only depend on the probability that z'_j is correct (which is the case at least 50% of the time). Thus, when *Z* again equals the sum of all Z_j, we just need to show that the probability that *Z* is less than $m/2$ (not a majority value) is less than 10% or $1/(10n)$. Since m=200n, this is the case by Chebychevs Inequality. Since *B* can invert *f(x)* in this way, we contradict the notion that *f* is a one-way permutation and thus we have proven that no algorithm can compute $x \odot r$ from *f(x)* and *r* alone more than 50% of the time.

5.5. Getting Arbitrarily Large Stretch

Now that we have pseudorandom generators whose output length is one greater than the input length, we now show pseudorandom generators whose output length is polynomial in terms of the input length n.

We assert that if *f* is a one-way permutation, then a function *G* is a pseudorandom generator for stretch *l(2n)=n+nc* if it maps binary strings *x* and *r* to outputs *r*, $f^l(x) \odot r$, $f^{l-1}(x) \odot r$, , $f^1(x) \odot r$ (where *l=nc* and f^n is *f* run *n* times on the input).

We can prove the above by showing that any bit of *f* cant be successfully predicted more than 50% of the time. Suppose for sake of contradiction that an algorithm *A* can predict with at least 50% success the ith bit of the string $f^i(x) \odot r$ given the outputs *r*, $f^l(x) \odot r$, $f^{l-1}(x) \odot r$, ..., $f^{i+1}(x) \odot r$. We now show that an algorithm *B* can determine *x* with more than 50% success (which would violate the Goldreich-Levin theorem) from inputs *r* and *y=f(x)*. Let *B* compute *a=A(r, $f^{l-i-1}(y) \odot r$, ..., f(y)$\odot$r, y$\odot$r)* for some random i. Let *x=f^i(x')* for some *x'* and run *A* to predict $f^i(x') \odot r$. From this predicted value, *B* can get $x \odot r$ which inverts *f* and makes it not one-way, a contradiction.

With this proven assertion that pseudorandom generators can securely generate a pseudorandom stretch polynomial in length in terms of an input length n, cryptosystems can encrypt long messages using only short keys.

6. ZERO KNOWLEDGE PROOF

An interactive proof system is an abstraction of a messaging system between two parties, the verifier and the prover, in order to ascertain whether or not a string belongs to a language.

A zero-knowledge proof is an interactive proof system where the prover aims to persuade the verifier that a statement is true without providing any additional knowledge other than the correctness of the proposition in question[Goldwasser et al. 1985; Goldreich and Oren 1994]. In order words, in zero-knowledge abstraction, while communicating about the correctness of a proposition, no secret information is revealed.

For instance, assume that there is a client-server application where the client needs authorization to login to the system. Hence, the client transmits its plaintext "username" and "password" to the server, then server computes the password hash and compares the stored value to the computed value before determining whether or not to let the client login to the system. However, after the client transmits its plaintext password to the server, server learns both the "username" and "password" of the client, and in the event that the server is compromised, all secrets will be revealed. The importance of zero-knowledge proofs may be grasped much easier when we consider wide usage of of credit card numbers, passwords, and all personal information in todays connected world [Feng and McMillin 2014].

A probabilistic polynomial-time algorithm is called zero-knowledge proof of L if the following three conditions hold:

— *Completeness*: the protocol accepts a true statement with a probability greater than 2/3.
— *Soundness*: the protocol accepts a false statement with a probability less than 1/3.
— *Perfect Zero-Knowledge*: There exists a simulator S such that an outside observer cannot distinguish between the execution of the simulator and the execution of the real prover.

6.1. Example - Quadratic Residue

Quadratic residue is one of the popular problems used in order to describe the zero-knowledge proof concept. A number n is a quadratic residue (mod n) if there is a number s such that $x = s^2$ *(mod n)*. Without knowing factorization of n, it is hard to tell whether x is a quadratic residue mod n or not. For instance, 1,3,4,9,10,12 are quadratic residues of 13.

Quadratic residue zero-knowledge protocol is designed as follows [Tompa 1988]:

Let $x=t^2$ *(mod N)*, and Prover knows x, t, and N, and Verifier knows x and N, and Prover wishes to persuade Verifier that s/he knows x, t and N and x is a quadratic residue without revealing any of these numbers:

— Prover picks a random r such that $y = r^2(modN)$, and sends residue y to Verifier.
— Verifier picks a random number $b \in \{0, 1\}$ and sends b to Prover.
— Prover sends $z = (t^b * r)(modN)$
— Verifier tests z^2
If $b = 0$, checks $z^2 == r^2(modN) = y$
If $b = 1$, checks $z^2 == (t^2 * r^2)(modN) = (t^2(modN))(r^2(modN))(modN) = xy(modN)$ (may be correct if and only if x and y are quadratic residues) (Note that : ab mod n = (((a mod n)(b mod n)) (mod n))).

If check passes, output ”yes”, otherwise output ”no”.

Proof:

— Completeness: If r is a quadratic residue and if the parties follow the protocol, then the verifier accepts with probability 1;
— Soundness: If r is not a quadratic residue, then the verifier says “no” with probability at least one half (i.e. when b=0). If r is a quadratic residue, but y is not a residue, then the verifier says “no” with probability at least one half (i.e when b=1).
— Perfect Zero-knowledge: Let S^* be a probabilistic polynomial time simulator running as a Verifier. Hence, first it sends a random number $b \in \{0, 1\}$ to the Prover. After receiving z^2, the Simulator S^* can verify in polynomial time, and output the result without learning any secret of the Prover. Even a cheating simulator S^* cannot learn any secret information. Details of Perfect Zero Knowledge proof are provided in [Santis et al. 1994].

7. APPLICATIONS

7.1. Tossing Coins over the Audio Phone

Assume that two friends A and B were talking over the phone, and one of them tossed a coin in order to make a decision. If A (or B) tossed the coin, and B (or A) made the guess, and told his guess, there was nothing to prevent the tossing party from lying about the result. Tossing a coin over the phone is a very good example of zero knowledge proof systems. In order to solve this problem, the following protocol may be used:

— A chooses $a \in \{0, 1\}$ and $r \in \{0, 1\}^n$ and computes a one-way permutation function $com(a, r)$, and sends $c = com(a, r)$ to B.
— B chooses $b \in \{0, 1\}$ and sends b to A.
— A sends a and r, and B checks if $c == com(a, r)$.
— Both compute $coin = a \oplus b$.

Proof:
— Completeness: If this is a honest protocol, and the parties follow the protocol, then the both parties accept with probability 1;
— Soundness: In this protocol B knows com function, and if a and r are provided, B can prove that $c == com(a, r)$, however B cannot guess a, without knowing a and r. Hence, A cannot bias the system and thus it is not possible to cheat the system.
— Perfect Zero-Knowledge: Let’s assume that there is a simulator S^* instead of B, and it sends a random $b \in \{0, 1\}$ to A. Since Simulator S^* does not know about a and r, it is not possible for A to guess a using com. After Simulator $S*$ receives a and r, it can verify that $c == com(a, r)$ is valid in polynomial time. Hence, A cannot update value a according to the value of b it receives, and Simulator S^* cannot guess a until it receives a and r. The decision is made is $a \oplus b$. Even a cheating simulator S^* cannot learn any secret information.

7.2. Secure Multi-party Computations

There are n parties wishing to compute a common function $f(x_1, x_2, ..., x_n)$ where each party holds a string $x_j \in \{0, 1\}^k$, and none of them can reveal its string. For instance, assume that we are building a system where the average grade of the class should be computed. However, a student’s privacy is important and it is not possible to keep

student's records in a database; rather, students need to input their scores when the average is computed. Hence we need to compute $avg(score_{s1}, score_{s2}, ..., score_{sn})$ without revealing individual scores. A simple solution with two servers may be provided as follows: each student si picks a random number $k \in [0, score_{si}]$ and sends k to $server_1$ and $score_{si} - k$ to $server_2$. Hence, none of the servers can reveal exact information, and each server adds the received score to its sum. Thus, the average can be computed as $\frac{\sum_{i=0}^{n} score_{si}^{server_1} + \sum_{i=0}^{n} score_{si}^{server_2}}{n}$.

8. SUMMARY AND FINAL REMARKS

In this paper, basic foundations of cryptography to encrypt and decrypt data and some sample applications were provided. Through our discussion, we explained why one-time pad is not a practical solution even if it gives perfect security. We also discussed the alternative ways that can be used to generate a secure key such as using one-way functions. The use of one-way functions allows pseudorandom generators to create long random-like strings from short truly random inputs so that secure keys can be shorter than the messages they encrypt. Such pseudorandom stretches are provably secure against polynomial-time adversaries due to proofs of unpredictability for each bit of the stretch. Finally, we discussed interactive zero-knowledge proof systems which not only ensure privacy of transmitted data from eavesdroppers but also provide a higher level of security by hiding secret data even from the receiver of the system, and we completed our discussion with some sample applications of zero-knowledge systems.

REFERENCES

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: a Modern Approach*. Cambridge University Press.

Manuel Blum and Silvio Micali. 1984. How to Generate Cryptographically Strong Sequences of Pseudo-random Bits. *SIAM J. Comput.* 13, 4 (Nov. 1984), 850–864.

Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael: AES-the Advanced Encryption Standard*. Springer Science & Business Media.

Li Feng and Bruce McMillin. 2014. A Survey on Zero-Knowledge Proofs. 94 (2014), 25–69.

Oded Goldreich. 2004. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA.

O. Goldreich and L. A. Levin. 1989. A Hard-core Predicate for All One-way Functions. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing (STOC '89)*. ACM, New York, NY, USA, 25–32.

O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC '87)*. ACM, New York, NY, USA, 218–229.

Oded Goldreich, Silvio Micali, and Avi Wigderson. 1991. Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-knowledge Proof Systems. *J. ACM* 38, 3 (July 1991), 690–728.

Oded Goldreich and Yair Oren. 1994. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology* 7, 1 (1994), 1–32.

Shafi Goldwasser and Silvio Micali. 1982. Probabilistic Encryption &Amp; How to Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC '82)*. ACM, New York, NY, USA, 365–377. DOI:http://dx.doi.org/10.1145/800070.802212

S Goldwasser, S Micali, and C Rackoff. 1985. The Knowledge Complexity of Interactive Proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC '85)*. ACM, New York, NY, USA, 291–304.

Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* 17, 2 (1988), 281–308.

Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. 1999. A Pseudorandom Generator from Any One-Way Function. *SIAM J. Comput.* 28, 4 (1999), 1364–1396.

Russell Impagliazzo and Michael Luby. 1989. One-Way Functions are Essential for Complexity Based Cryptography. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*. IEEE, 230–235.

Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography*. CRC Press.

A. De Santis, G. Di Crescenzo, and G. Persiano. 1994. Secret Sharing and Perfect Zero Knowledge. Springer–Verlag, 73–84.

Adi Shamir. 1983. On the Generation of Cryptographically Strong Pseudorandom Sequences. *ACM Trans. Comput. Syst.* 1, 1 (Feb. 1983), 38–44.

Michael Sipser. 2006. *Introduction to the Theory of Computation*. Vol. 2. Thomson Course Technology Boston.

Martin Tompa. 1988. Zero Knowledge Interactive Proofs of Knowledge (A Digest). In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*. Morgan Kaufmann Publishers Inc., 1–12.

Andrew C. Yao. 1982a. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS '82)*. IEEE Computer Society, Washington, DC, USA, 160–164.

Andrew C. Yao. 1982b. Theory and Application of Trapdoor Functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS '82)*. IEEE Computer Society, Washington, DC, USA, 80–91.